

A decomposition method with minimum communication amount for parallelization of multi-dimensional FFTs

Truong Vinh Truong Duy^{a,b,*}, Taisuke Ozaki^a

^a*Research Center for Simulation Science, Japan Advanced Institute of Science and Technology (JAIST), 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan*

^b*Institute for Solid State Physics, The University of Tokyo, Kashiwanoha 5-1-5, Kashiwa, Chiba 277-8581, Japan*

Abstract

The fast Fourier transform (FFT) is undoubtedly an essential primitive that has been applied in various fields of science and engineering. In this paper, we present a decomposition method for parallelization of multi-dimensional FFTs with smallest communication amount for all ranges of the number of processes compared to previously proposed methods. This is achieved by two distinguishing features: adaptive decomposition and transpose order awareness. In the proposed method, the FFT data are decomposed based on a row-wise basis that maps the multi-dimensional data into one-dimensional data, and translates the corresponding coordinates from multi-dimensions into one-dimension so that the resultant one-dimensional data can be divided and allocated equally to the processes. As a result, differently from previous works that have the dimensions of decomposition pre-defined, our method can adaptively decompose the FFT data on the lowest possible dimensions depending on the number of processes. In addition, this row-wise decomposition provides plenty of alternatives in data transpose, and different transpose order results in different amount of communication. We identify the best transpose orders with smallest communication amounts for the 3-D, 4-D, and 5-D FFTs by analyzing all possible cases. Given both communication efficiency and scalability, our method is promising in development of highly efficient parallel packages for the FFT.

*Corresponding author. Tel.: +81 761 51 1987

Email addresses: `duyvt@jaist.ac.jp, issp.u-tokyo.ac.jp` (Truong Vinh Truong Duy), `t-ozaki@jaist.ac.jp` (Taisuke Ozaki)

Keywords: Fast Fourier transform; Multi-dimensional FFTs; Domain decomposition for FFTs; Domain decomposition method; Parallel FFTs

1. Introduction

Since the pioneering work of J. W. Cooley and J. W. Tukey in 1965 [1], the fast Fourier transform (FFT) has become an essential kernel for numerous fields of science and engineering, such as electronic structure calculations, digital signal processing, medical image processing, communications, astronomy, geology, optics, and so on [2, 3, 4, 5]. In those applications, the FFT is usually performed with a large data set in multiple dimensions many times, making the FFT a computationally expensive calculation. Given the importance of the FFT and widespread of massively parallel supercomputers of multi-core processors, it should come as no surprise that there have been a lot of efforts to parallelize the FFT that can be roughly categorized into two approaches: the direct approach for one-dimensional (1-D) FFTs, and the transpose approach for multi-dimensional FFTs. In the former approach, the 1-D FFT is parallelized by dividing the original problem into sub-problems recursively, which are assigned to the processes for solving in parallel [6, 7, 8].

In the latter approach for multi-dimensional FFTs, the FFT data are delivered to the processes so that they can have enough data to perform the sequential 1-D FFTs locally in one specific dimension in parallel, for example with the wildly popular FFTW package [9]. The primary concern of this paper is the communication efficiency of the latter approach, as the data must be transposed, repeatedly if necessary until the data for all the dimensions have been FFT-transformed, for conducting the 1-D FFTs in other dimensions. The data transpose requires communications among the processes, and must be considered carefully, whereas the 1-D FFT is independent and communication-free. As a result, the communication efficiency heavily depends on the domain decomposition method, which should minimize the amount of communication, while not sacrificing scalability.

The domain decomposition can be carried out in any degree from one- to M -dimensions for the M -dimensional (M -D) FFT, facing the trade-off between communication efficiency and scalability. The lower the degree of the domain decomposition is, the higher the communication efficiency is, and the lower the scalability is. That said, the 1-D decomposition achieves the highest communication efficiency with the lowest scalability, in contrast to

the M -D decomposition, which carries the lowest communication efficiency with the highest scalability.

Let us take the most popular three-dimensional (3-D) FFT as an example, where the domain decomposition method exists in three forms: one-, two-, and three-dimensions. The alphabet hereafter is used to denote the dimensions, for example, a for the first dimension, b for the second dimension, etc. The 1-D decomposition [2, 10], or slab decomposition, divides the 3-D data into equal blocks of complete ab -planes, for instance, along the c -axis, and requires only one transpose step with the smallest amount of communication, but the number of processes applicable is limited to the size of one dimension. The 3-D decomposition, or volumetric decomposition, partitions the 3-D data along all three dimensions [11, 12, 13, 14], and therefore three data transpose steps are necessary to perform the 1-D FFTs along the three dimensions, making the 3-D decomposition the costliest in terms of communication amount, yet the most scalable, as the maximum number of processes is equal to the size of the 3-D data. Sandwiched between the 1-D and 3-D decompositions is the 2-D decomposition (pencil or rod decomposition) [15, 16, 17, 18] that draws smaller amount of communication than the 3-D decomposition, and offers higher scalability than the 1-D decomposition, as the number of processes applicable is now up to the size of 2-D plane.

Our work is motivated by the fact that the aforementioned domain decomposition methods usually pre-define the dimensions of decomposition, regardless of the number of processes. In particular, the 2-D decomposition partitions in two dimensions, even when the number of processes is less than the size of one dimension and it is possible to decompose in only one dimension to reduce the communication amount, because lower degree of decomposition leads to smaller amount of communication. As can be expected, the situation is worse for the 3-D decomposition, which works in three dimensions paying no attention to the number of processes. For better communication efficiency, this factor should be taken into account in the decomposition method.

In addition, we observe that the order of data transpose has no effect upon the communication efficiency in those decomposition methods, as they treat the dimensions involved in the decomposition in the same way. For example, as mentioned above, the 2-D decomposition divides the ab -plane evenly, i.e., the a - and b -axes are the same and the processes are allocated approximately equal numbers of data points along these two axes. In the two subsequent transpose steps, there is no difference in doing ac or bc first,

because they all result in the same amount of communication. However, we find that if the dimensions involved in the decomposition are handled differently, specifically the data points on the ab -plane are divided to the processes in ascending order of their a - and then b -coordinates, the transpose order will become a key factor in the subsequent transpose steps. Choosing the right transpose order will actually reuse a lot of data from the previous step and consequently reduce a substantial communication amount.

Another motivation is that while the decomposition method for parallel 3-D FFTs has been extensively investigated, little work has explored beyond them so far. In fact, 4-D and 5-D FFTs have various applications, and examples can be found in fields such as lattice quantum chromodynamics simulations, where a Landau gauge fixing algorithm is implemented using the 4-D FFT [19], in medical image processing with a 4-D FFT-based filtering [20], in photography [21, 22], in drug design with 5-D FFT-based protein-protein docking algorithms [23, 24, 25], and others [26]. Hence, there is a real need to develop parallel M -D FFTs beyond 3-D FFTs.

In this paper, we present a decomposition method for parallelization of multi-dimensional FFTs with two distinguishing features: adaptive decomposition and transpose order awareness for achieving smallest communication amount compared to previously proposed methods. In our method, the FFT data are decomposed based on a row-wise basis that maps the M -D data into 1-D data, and translates the corresponding coordinates from multi-dimensions into one-dimension for equally dividing and allocating the resultant 1-D data to the processes. As a result, our method can adaptively decompose the FFT data on the lowest possible dimensions depending on the number of processes so that the communication amount can be minimized in the first place, differently from previous works that have fixed-dimensions of decomposition. In particular, the method decomposes in one dimension if the number of processes is less than or equal to the size of one dimension, in two dimensions if the number of processes is greater than the size of one dimension and less than or equal to the size of two dimensions, up to M -dimensions. Another unique feature of our method is the awareness of the transpose order. The row-wise decomposition provides plenty of alternatives in data transpose, and different transpose order results in different amount of communication. By analyzing all possible cases for transpose order, we find out the best transpose orders with smallest communication amounts for 3-D, 4-D, and 5-D FFTs. Finally, our method is generalized to M -D FFTs.

The remainder of the paper is organized as follows. Section 2 describes

M -D FFTs. The domain decomposition method is presented in Section 3, and Section 4 gives comparison in terms of communication amount between our method and other methods. Our study is concluded in Section 5.

2. Multi-dimensional FFTs

We start the discussion by introducing the M -D FFT by way of the 1-D FFT. The 1-D FFT transforms a 1-D data $X(j)$ of N complex numbers ($X(0), X(1), \dots, X(N-1)$) into another 1-D data $\bar{X}(k)$ of N complex numbers ($\bar{X}(0), \bar{X}(1), \dots, \bar{X}(N-1)$) as follows:

$$\bar{X}(k) = \sum_{j=0}^{N-1} \omega_N^{jk} X(j), \quad (1)$$

where $\omega_N = e^{-2\pi i/N}$, $i = \sqrt{-1}$, and the factor is dropped for simplicity.

The 1-D FFT transforms the 1-D data that has exactly one discrete variable j into another 1-D data structure. Similarly, the M -D FFT transforms an M -D data $X(j_1, j_2, \dots, j_M)$ that has M discrete variables j_1, j_2, \dots, j_M into another M -D data $\bar{X}(k_1, k_2, \dots, k_M)$, defined by

$$\bar{X}(k_1, k_2, \dots, k_M) = \sum_{j_1=0}^{N_1-1} \left(\omega_{N_1}^{j_1 k_1} \sum_{j_2=0}^{N_2-1} \left(\omega_{N_2}^{j_2 k_2} \dots \sum_{j_M=0}^{N_M-1} \omega_{N_M}^{j_M k_M} X(j_1, j_2, \dots, j_M) \right) \right), \quad (2)$$

where $\omega_{N_r} = e^{-2\pi i/N_r}$, and $1 \leq r \leq M$.

As can be seen from the above equation, the M -D FFT can be computed in M single steps, with each performing the 1-D FFTs along one specific dimension. This is a crucial feature exploited by the domain decomposition method for parallelization of the M -D FFT. The first step conducts the 1-D FFTs along the last dimension, for instance, followed by a transpose operation. The second step executes the 1-D FFTs along the dimension prior to the last dimension, and so on. Lastly, the M th step carries out the 1-D FFTs along the first dimension, ending the M -D FFT.

3. Domain decomposition method

In this section, we present our domain decomposition method, starting with the 3-D FFT for ease of understanding. We then provide a general description of the method for the M -D FFT, and finally we describe the method for the 4-D and 5-D FFTs, and beyond them.

3.1. 3-D FFTs

Here we illustrate and examine our decomposition method for the 3-D FFT. Assume that the numbers of data points along the a -, b -, and c -axes are N_1 , N_2 , and N_3 , respectively, the number of processes is N_p , and $myid$ is the process identification.

Our method is basically based on a row-wise decomposition. It first translates the original 3-D FFT data $X_{3D}(a, b, c)$ into a 1-D data $X_{1D}(x)$, as illustrated in Fig. 1 for the abc order as an example in a total of $3 \times 2 \times 1 = 6$ orders. The relationship between a 3-D coordinate $X_{3D}(a_1, b_1, c_1)$ and a 1-D coordinate $X_{1D}(x_1)$ in the abc order is given by

$$x_1 = a_1 \times N_2 \times N_3 + b_1 \times N_3 + c_1. \quad (3)$$

It is worth mentioning that the relationships in other orders different from abc can be derived in the same way.

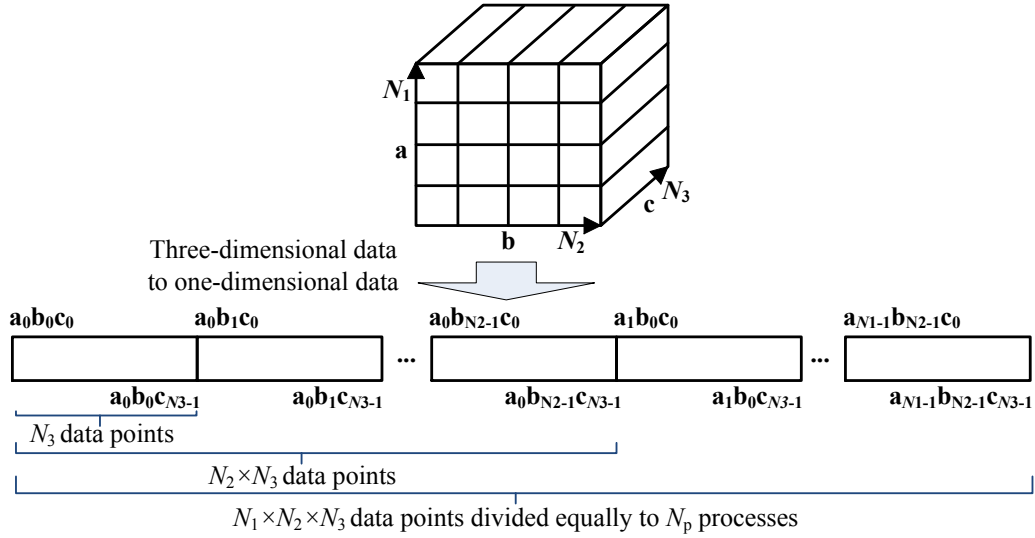


Figure 1: 3-D FFTs: 3-D data to 1-D data mapping in row-wise decomposition for the abc order.

In order to identify the starting and ending points allocated to each process in the domain decomposition determined by N_p , $myid$, N_1 , N_2 , and N_3 , we define a function $f_{3D}()$ for translating the 3-D and 1-D coordinates as

follows:

$$f_{3D}(N_1, N_2, N_3, N_p, myid) = \begin{cases} \left\lfloor \frac{N_1 \times myid}{N_p} \right\rfloor \times N_2 N_3, & \text{if } N_p \leq N_1; \\ \left\lfloor \frac{N_1 N_2 \times myid}{N_p} \right\rfloor \times N_3, & \text{if } N_1 < N_p \leq N_1 N_2; \\ \left\lfloor \frac{N_1 N_2 N_3 \times myid}{N_p} \right\rfloor, & \text{if } N_1 N_2 < N_p \leq N_1 N_2 N_3, \end{cases} \quad (4)$$

where $\lfloor \cdot \rfloor$ is the floor function. Our method then equally divides the resultant 1-D data to the processes, in which a process with $myid$ is assigned the data points from $X_{1D}(x_{myid}^s)$ to $X_{1D}(x_{myid}^e)$ in one dimension, where

$$x_{myid}^s = f_{3D}(N_1, N_2, N_3, N_p, myid), \quad (5)$$

$$x_{myid}^e = f_{3D}(N_1, N_2, N_3, N_p, myid + 1) - 1. \quad (6)$$

These 1-D coordinates can be translated back to the 3-D ones to obtain the corresponding starting and ending coordinates in three dimensions as

$$a_{myid}^{(s,e)} = \left\lfloor \frac{x_{myid}^{(s,e)}}{N_2 N_3} \right\rfloor, \quad (7)$$

$$b_{myid}^{(s,e)} = \left\lfloor \frac{x_{myid}^{(s,e)} - a_{myid}^{(s,e)} N_2 N_3}{N_3} \right\rfloor, \quad (8)$$

$$c_{myid}^{(s,e)} = x_{myid}^{(s,e)} - a_{myid}^{(s,e)} N_2 N_3 - b_{myid}^{(s,e)} N_3, \quad (9)$$

where $X_{3D}(a_{myid}^s, b_{myid}^s, c_{myid}^s)$ and $X_{3D}(a_{myid}^e, b_{myid}^e, c_{myid}^e)$ are the starting and ending points, respectively, in three dimensions for a process with $myid$.

Consequently, the decomposition has three forms depending on the number of processes. The distribution of data points is carried out in 1-D, 2-D, or 3-D data defined by the first one, two, or three dimensions, respectively. For example, with N_p processes and $N_1 < N_p \leq N_1 N_2$, the decomposition in the abc order takes place in the 2-D decomposition, where the data points on the ab -plane with the c -axis are divided over the processes in ascending order of their a - and b -coordinates so that each process has approximately the same number of data points on the ab -plane. The data points extending along the c -axis that have the same a - and b -coordinates are also assigned

to the same process. Therefore with the 2-D decomposition on the ab -plane in the abc order, a process with $myid$ will be assigned the data points from $X_{3D}(a_{myid}^s, b_{myid}^s, 0)$ to $X_{3D}(a_{myid}^e, b_{myid}^e, N_3 - 1)$ in ascending order of the a -, b -, and c -coordinates, where a_{myid}^s , b_{myid}^s , a_{myid}^e , and b_{myid}^e can be obtained from Eqs. 7 and 8, with

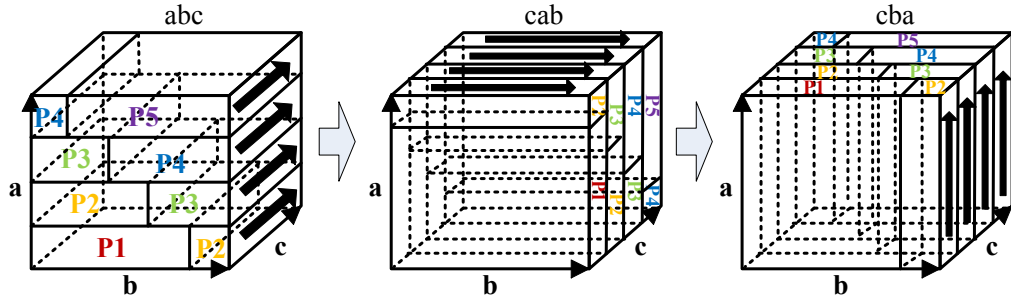
$$x_{myid}^s = \left\lfloor \frac{N_1 N_2 \times myid}{N_p} \right\rfloor \times N_3, \quad (10)$$

$$x_{myid}^e = \left\lfloor \frac{N_1 N_2 \times (myid + 1)}{N_p} \right\rfloor \times N_3 - 1. \quad (11)$$

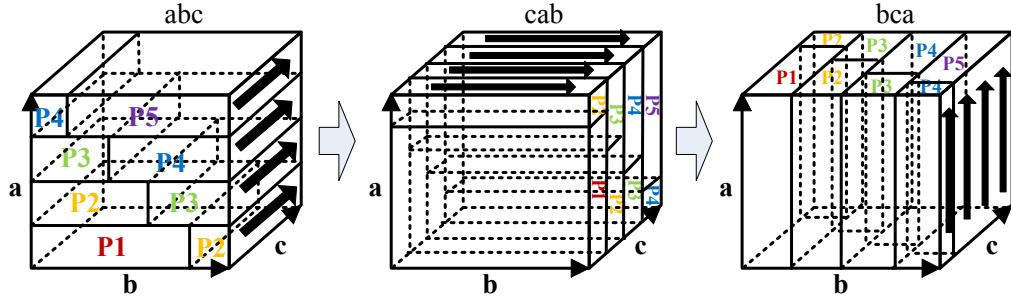
In subsequent transpose steps, the decomposition can occur in any order of combination of the three dimensions a , b , and c .

Figure 2 exemplifies the operation of our method with transpose-order awareness (a) and transpose-order unawareness (b), followed by a conventional 2-D method for comparison (c). The transpose order in Fig. 2(a) is $abc \rightarrow cab \rightarrow cba$, and in Fig. 2(b) $abc \rightarrow cab \rightarrow bca$. Even though the only difference between them is the last decomposition, cba as against bca , this has far-reaching implications for the amount of reused data, because a majority of data can be reused with the transpose from cab to cba , while the transpose from cab to bca leaves only a minority of data that can be reused. For instance, with process P1, Fig. 2(a) shows a large overlap between the areas assigned to it in cab and cba , implying that a large amount of data can be reused, whereas the overlap between cab and bca is small (Fig. 2(b)). In fact, as revealed later in Fig. 3(a), the amount of communication with transpose-order unawareness, $abc \rightarrow cab \rightarrow bca$, is doubled compared to transpose-order awareness, $abc \rightarrow cab \rightarrow cba$. On the other hand, the conventional 2-D decomposition is applied to two dimensions that are treated in the same way so that the processes have approximately equal numbers of data points on these two dimensions, leaving no difference between cba and bca , and eventually no effect of the transpose order. Also, though the illustrations are intended for 2-D decomposition, the extension to 1-D and 3-D decompositions is straightforward.

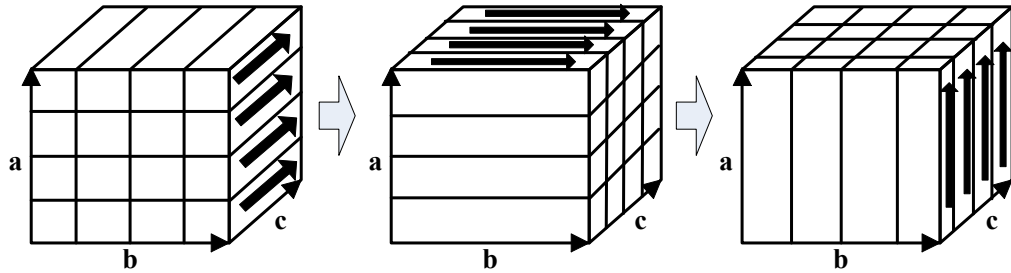
Figure 3(a) shows a full analysis on the amount of communication to the number of processes with all 8 possible cases of transpose order for the 3-D FFT, with $N \times N \times N$ data points for the sake of simplicity and clear presentation. Interestingly, the amount of communication is grouped into only two patterns of communication in our method, and there is one pattern



(a) Transpose-order awareness: transpose from cab to cba , a majority of data can be reused.



(b) Transpose-order unawareness: transpose from cab to bca , only a minority of data can be reused. The amount of communication is doubled compared to (a).

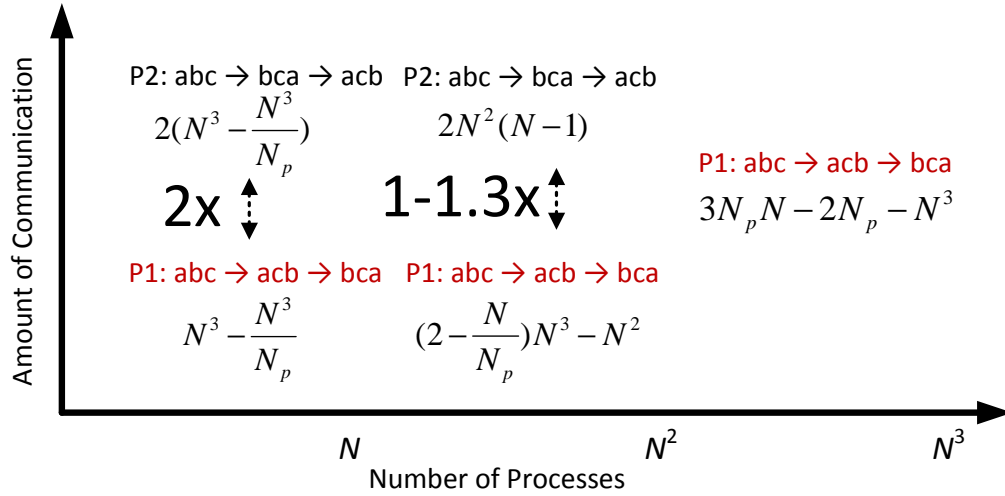


(c) Conventional 2-D decomposition: all dimensions are the same.

Figure 2: 3-D FFTs with 2-D decomposition: our method with transpose-order awareness (a) and unawareness (b), and regular 2-D method (c).

| | | | | | | | | |
|---|------------------------|--------------------|--------------------|--------------------|------------------------|--------------------|------------------------|------------------------|
| | abc | | | | | | | |
| | cba | | bca | | cab | | acb | |
| | cab | acb | acb | cab | cba | bca | cba | bca |
| N_p : Number of processes $N \times N \times N$ grid | | | | | | | | |
| $N_p \leq N$ | $N^3 - N^3/N_p$ | $2(N^3 - N^3/N_p)$ | $2(N^3 - N^3/N_p)$ | $2(N^3 - N^3/N_p)$ | $N^3 - N^3/N_p$ | $2(N^3 - N^3/N_p)$ | $N^3 - N^3/N_p$ | $N^3 - N^3/N_p$ |
| $N < N_p < N^2$ | $(2 - N/N_p)N^3 - N^2$ | $2N^2(N-1)$ | $2N^2(N-1)$ | $2N^2(N-1)$ | $(2 - N/N_p)N^3 - N^2$ | $2N^2(N-1)$ | $(2 - N/N_p)N^3 - N^2$ | $(2 - N/N_p)N^3 - N^2$ |
| $N_p = N^2$ | $2N^2(N-1)$ | $2N^2(N-1)$ | $2N^2(N-1)$ | $2N^2(N-1)$ | $2N^2(N-1)$ | $2N^2(N-1)$ | $2N^2(N-1)$ | $2N^2(N-1)$ |
| $N^2 < N_p < N^3$ | $N^3 + 2N_p(N-1)$ | $N^3 + 2N_p(N-1)$ | $N^3 + 2N_p(N-1)$ | $N^3 + 2N_p(N-1)$ | $N^3 + 2N_p(N-1)$ | $N^3 + 2N_p(N-1)$ | $N^3 + 2N_p(N-1)$ | $N^3 + 2N_p(N-1)$ |
| $N_p = N^3$ | $3N^3(N-1)$ | $3N^3(N-1)$ | $3N^3(N-1)$ | $3N^3(N-1)$ | $3N^3(N-1)$ | $3N^3(N-1)$ | $3N^3(N-1)$ | $3N^3(N-1)$ |

(a) 8 transpose order cases and the amount of communication corresponding to the number of processes.



(b) 2 patterns (P1 and P2), the amount of communication corresponding to the number of processes, and the difference between their amount.

Figure 3: 3-D FFTs: all 8 cases (a), categorized into 2 patterns (b).

actually up to twice better than the other, as shown in Fig. 3(b). The difference between these two patterns is twice, when the number of processes is up to the size of one dimension, and up to 1.3 times, when the number of processes is in between the sizes of one and two dimensions. The difference remains almost unchanged by N . We find four transpose orders leading to the pattern with the smaller amount of communication for all ranges of the number of processes:

$$\begin{aligned} abc &\rightarrow acb \rightarrow bca, \\ abc &\rightarrow acb \rightarrow cba, \\ abc &\rightarrow cba \rightarrow cab, \\ abc &\rightarrow cab \rightarrow cba. \end{aligned}$$

In general, we can follow any of these four orders in parallelization of the 3-D FFT. In fact, our previous work has employed the method with the 2-D decomposition and the transpose order of $abc \rightarrow cab \rightarrow cba$ [27].

3.2. General description of the method

Let $X_{M-D}(N_1, N_2, \dots, N_M)$ be the M -D input data. Each process is allocated approximately $\frac{N_1 \times N_2 \times \dots \times N_M}{N_p}$ data points. The decomposition starts from the first dimension and gradually moves down to the last dimension of the data structure to assign the data points to the processes according to the number of processes. The problem turns to finding the starting and ending coordinates of each dimension of the data structure for each process, specifically, the starting point $X_{M-D}(x_{1,myid}^s, x_{2,myid}^s, \dots, x_{M,myid}^s)$ and the ending point $X_{M-D}(x_{1,myid}^e, x_{2,myid}^e, \dots, x_{M,myid}^e)$ that together define the data points distributed to a particular process with $myid$. This procedure can be generalized from the previous case of 3-D FFTs.

After the M -D input data have been translated into the 1-D data, a 1-D coordinate $X_{1D}(X_1)$ is identified from the equivalent M -D coordinate $X_{M-D}(x_1, x_2, \dots, x_M)$ as follows:

$$X_1 = x_1 \times N_2 \times N_3 \times \dots \times N_M + x_2 \times N_3 \times N_4 \times \dots \times N_M + \dots + x_{M-1} \times N_M + x_M. \quad (12)$$

As well as the case of the 3-D FFT, a function $f_{M-D}()$ for translating the M -D and 1-D coordinates is defined as

$$f_{M-D}(N_1, N_2, \dots, N_M, N_p, myid) = \begin{cases} \left\lfloor \frac{N_1 \times myid}{N_p} \right\rfloor \times N_2 N_3 \times \dots \times N_M, & \text{if } N_p \leq N_1; \\ \left\lfloor \frac{N_1 N_2 \times myid}{N_p} \right\rfloor \times N_3 \times \dots \times N_M, & \text{if } N_1 < N_p \leq N_1 N_2; \\ \dots & \\ \left\lfloor \frac{N_1 N_2 \times \dots \times N_M \times myid}{N_p} \right\rfloor, & \text{if } N_1 N_2 \times \dots \times N_{M-1} < N_p \leq N_1 N_2 \times \dots \times N_M. \end{cases} \quad (13)$$

In one-dimension, the data points from $X_{1D}(x_{myid}^s)$ to $X_{1D}(x_{myid}^e)$ are allocated to a process with $myid$, where

$$x_{myid}^s = f_{M-D}(N_1, N_2, \dots, N_M, N_p, myid), \quad (14)$$

$$x_{myid}^e = f_{M-D}(N_1, N_2, \dots, N_M, N_p, myid + 1) - 1. \quad (15)$$

Finally, in M -dimensions, the starting point $X_{M-D}(x_{1,myid}^s, x_{2,myid}^s, \dots, x_{M,myid}^s)$ and the ending point $X_{M-D}(x_{1,myid}^e, x_{2,myid}^e, \dots, x_{M,myid}^e)$ of a process with $myid$ are given by

$$x_{1,myid}^{(s,e)} = \left\lfloor \frac{x_{myid}^{(s,e)}}{N_2 N_3 \times \dots \times N_M} \right\rfloor, \quad (16)$$

$$x_{2,myid}^{(s,e)} = \left\lfloor \frac{x_{myid}^{(s,e)} - x_{1,myid}^{(s,e)} N_2 N_3 \times \dots \times N_M}{N_3 \times \dots \times N_M} \right\rfloor, \quad (17)$$

...

$$x_{M-1,myid}^{(s,e)} = \left\lfloor \frac{x_{myid}^{(s,e)} - x_{1,myid}^{(s,e)} N_2 N_3 \times \dots \times N_M - x_{2,myid}^{(s,e)} N_3 \times \dots \times N_M - \dots - x_{M-2,myid}^{(s,e)} N_{M-1} N_M}{N_M} \right\rfloor, \quad (18)$$

$$x_{M,myid}^{(s,e)} = x_{myid}^{(s,e)} - x_{1,myid}^{(s,e)} N_2 N_3 \times \dots \times N_M - x_{2,myid}^{(s,e)} N_3 \times \dots \times N_M - \dots - x_{M-1,myid}^{(s,e)} \times N_M. \quad (19)$$

With the definition of the function $f_{M-D}()$, the row-wise decomposition realizes the first feature of our method, adaptive decomposition. This is how our method is adaptive and flexible to the number of processes N_p . When N_p is less than or equal to the size of the first dimension N_1 , the method will partition the M -D data in only that dimension, and assign about $\frac{N_1}{N_p} \times (N_2 \times \dots \times N_M)$ data points to each process. Likewise, if N_p is between N_1 and $N_1 \times N_2$, the decomposition will take place in the first and second dimensions, and distribute approximately $\frac{N_1 \times N_2}{N_p} \times (N_3 \times \dots \times N_M)$ data points to each process. As lower degree of decomposition requires smaller amount of communication, our method is able to minimize the communication amount in the first place.

This adaptive decomposition is conducted when data transpose is performed to re-allocate the data points to the processes. The computation is relatively simple, and therefore its computational time is trivial. Figure 4 outlines the parallel M -D FFT with the decomposition for data transpose. The order of transpose used in the figure is general, and the actual order, accompanied by the amount of communication, is analyzed for the 3-D, 4-D, and 5-D FFTs.

Since this is a row-wise-based distribution, the method leaves a number of order options to be explored, because in this case, abc is no longer identical to cab with the 3-D FFT, as illustrated previously. The order of transpose plays a key role in the ability of re-using data that is directly related to the amount of communication. Let us calculate the total number of cases for the M -D FFT. Since we start with the first transpose, it has only a single case. In the next transpose, as we have $M - 1$ remaining dimensions to choose from and each dimension has $(M - 1)!$ cases to be explored, there are $(M - 1) \times (M - 1)!$ cases in this step. Similarly, there are $(M - 2) \times (M - 1)!$ cases in the third transpose, because we have $M - 2$ remaining dimensions with each dimension having $(M - 1)!$ cases. In the M th (final) transpose, there are $1 \times (M - 1)!$ cases. Multiplication of all the cases in M steps gives

- Input:** $X(N_1, N_2, \dots, N_M), N_p, myid$
Output: $\bar{X}(N_1, N_2, \dots, N_M)$
- 1 Step 1: Perform $N_1 \times N_2 \times \dots \times N_{M-1}$ 1-D FFTs, each with N_M points along the M th dimension.
 $\bar{X}^1(j_1, j_2, \dots, j_{M-1}, k_M) = \sum_{j_M=0}^{N_M-1} \omega_{N_M}^{j_M k_M} X(j_1, j_2, \dots, j_M)$
 - 2 Transpose: Conduct the adaptive decomposition.
 $\bar{X}^2(k_M, j_1, j_2, \dots, j_{M-1}) = \bar{X}^1(j_1, j_2, \dots, j_{M-1}, k_M)$
 - 3 Step 2: Perform $N_M \times N_1 \times N_2 \times \dots \times N_{M-2}$ 1-D FFTs, each with N_{M-1} points along the $(M-1)$ th dimension.
 $\bar{X}^3(k_M, j_1, j_2, \dots, j_{M-2}, k_{M-1}) = \sum_{j_{M-1}=0}^{N_{M-1}-1} \omega_{N_{M-1}}^{j_{M-1} k_{M-1}} \bar{X}^2(k_M, j_1, j_2, \dots, j_{M-1})$
 - 4 Transpose: Conduct the adaptive decomposition.
 $\bar{X}^4(k_M, k_{M-1}, j_1, j_2, \dots, j_{M-2}) = \bar{X}^3(k_M, j_1, j_2, \dots, j_{M-2}, k_{M-1})$
 - 5 ...
 - 6 Step M : Perform $N_M \times N_{M-1} \times \dots \times N_2$ 1-D FFTs, each with N_1 points along the first dimension.
 $\bar{X}^{2M-1}(k_M, k_{M-1}, \dots, k_2, k_1) = \sum_{j_1=0}^{N_1-1} \omega_{N_1}^{j_1 k_1} \bar{X}^{2M-2}(k_M, k_{M-1}, \dots, k_2, j_1)$
 - 7 Transpose: Conduct the adaptive decomposition.
 $\bar{X}(k_1, k_2, \dots, k_{M-1}, k_M) = \bar{X}^{2M-1}(k_M, k_{M-1}, \dots, k_2, k_1)$

Figure 4: Parallel M -D FFTs with adaptive decomposition for data transpose.

us the total number of cases for the M -D FFT

$$\begin{aligned}
C_{M-D} &= \underbrace{1}_{\text{1st transpose}} \times \underbrace{(M-1) \times (M-1)!}_{\text{2nd transpose}} \times \underbrace{(M-2) \times (M-1)!}_{\text{3rd transpose}} \times \dots \\
&\times \underbrace{2 \times (M-1)!}_{(M-1)\text{th transpose}} \times \underbrace{1 \times (M-1)!}_{M\text{th transpose}} \\
&= (M-1) \times (M-2) \times \dots \times 2 \times 1 \times (M-1)!^{M-1} \\
&= (M-1)!^M
\end{aligned} \tag{20}$$

Table 1 shows the number of order cases corresponding to the number of dimensions. With the 3-D FFT, there are 8 cases only, which can be examined manually. However, with the 4-D and 5-D FFTs, these numbers are 1,296 and 7,962,624, respectively, that can be investigated thoroughly by computer simulations. Even so, it is practically difficult with 2,985,984,000,000 cases for the 6-D FFT. Therefore, we have no choice but to limit ourselves to the

Table 1: Number of cases.

| Number of dimensions | Number of cases |
|----------------------|-------------------|
| 2 | 1 |
| 3 | 8 |
| 4 | 1,296 |
| 5 | 7,962,624 |
| 6 | 2,985,984,000,000 |
| M | $(M - 1)!^M$ |

5-D FFT, and try to generalize the results to those beyond them.

3.3. 4-D FFTs, 5-D FFTs, and beyond

Figure 5 depicts the patterns, the amount of communication corresponding to the number of processes, and the difference between their amount in different ranges for the 4-D FFT (a) and 5-D FFT (b), where the M -D data are assumed to have an equal size in all dimensions. Due to their complex nature, we do not have any illustrations for the decomposition method. As it is also difficult to present all possible cases, 1,296 for the 4-D FFT and 7,962,624 for the 5-D FFT, here we only show the best and worst patterns.

With the 4-D FFT, we find four transpose orders:

$$abcd \rightarrow abdc \rightarrow dcba \rightarrow dcab,$$

$$abcd \rightarrow abdc \rightarrow dcab \rightarrow dcba,$$

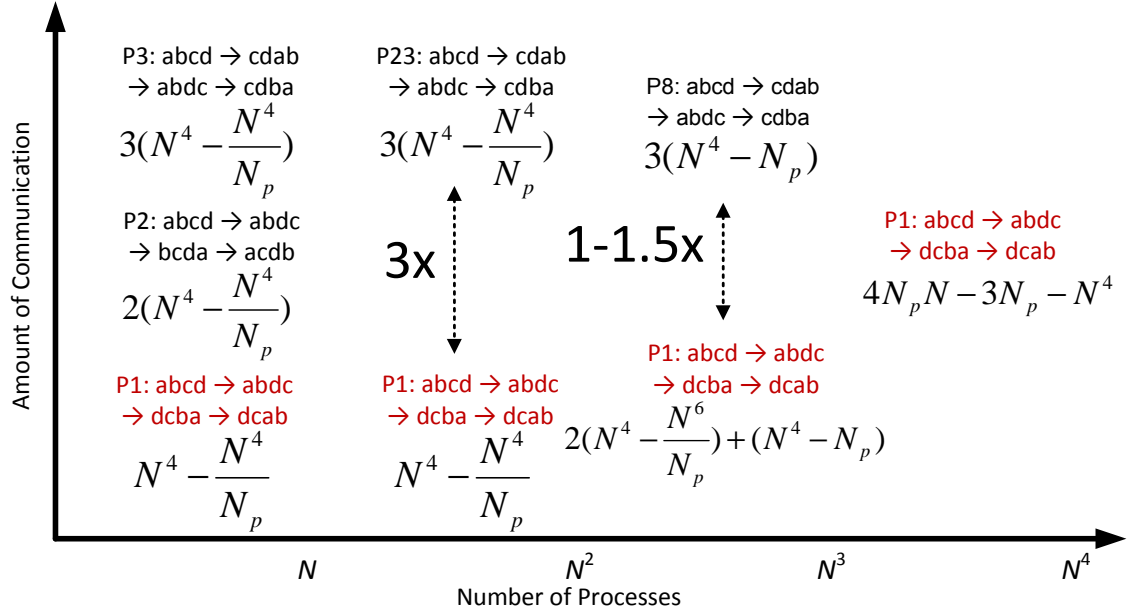
$$abcd \rightarrow abdc \rightarrow cdab \rightarrow cdba,$$

$$abcd \rightarrow abdc \rightarrow cdba \rightarrow cdab,$$

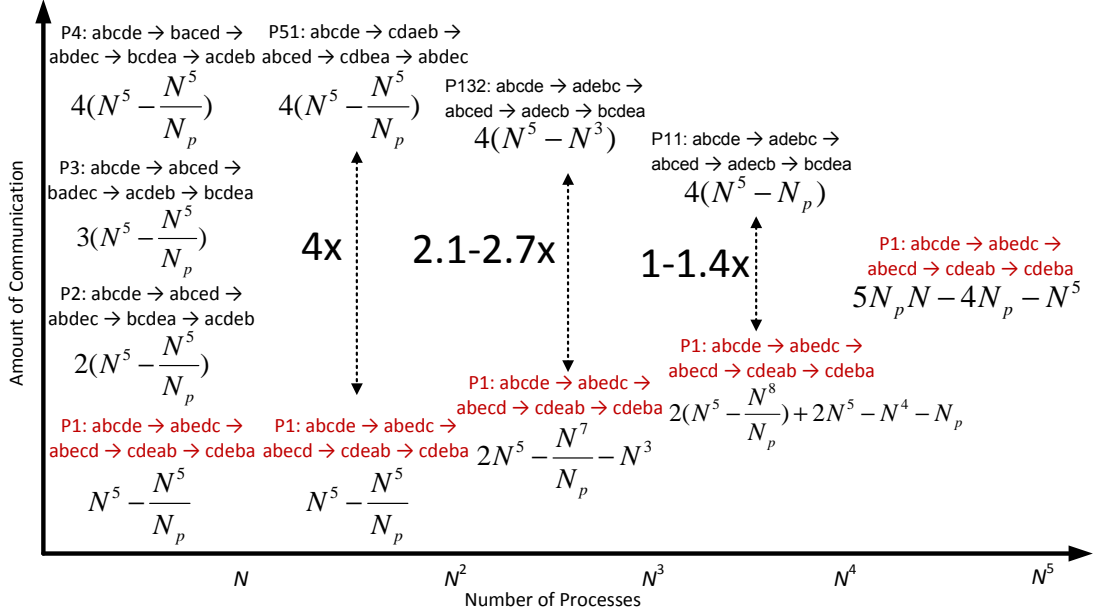
that always offer the smallest amount of communication for all ranges of the number of processes. The best transpose order is exactly 3 times better than the worst order for up to N^2 processes, and up to 1.5 times for the next range of $[N^2, N^3]$. The difference remains almost the same, apparently unaffected by N .

The number of always-best transpose orders is found to be higher with the 5-D FFT than with the 4-D FFT: 96 orders, including

$$abcde \rightarrow abced \rightarrow abedc \rightarrow cedba \rightarrow cedab,$$



(a) 4-D FFTs: the patterns, the amount of communication corresponding to the number of processes, and the difference between their amount.



(b) 5-D FFTs: the patterns, the amount of communication corresponding to the number of processes, and the difference between their amount.

Figure 5: 4-D and 5-D FFTs.

$$\begin{aligned}
abcde &\rightarrow abced \rightarrow abedc \rightarrow ecdba \rightarrow ecdab, \\
abcde &\rightarrow abced \rightarrow abedc \rightarrow cdeba \rightarrow cdeab, \\
abcde &\rightarrow abced \rightarrow abedc \rightarrow cdeab \rightarrow cdeba.
\end{aligned}$$

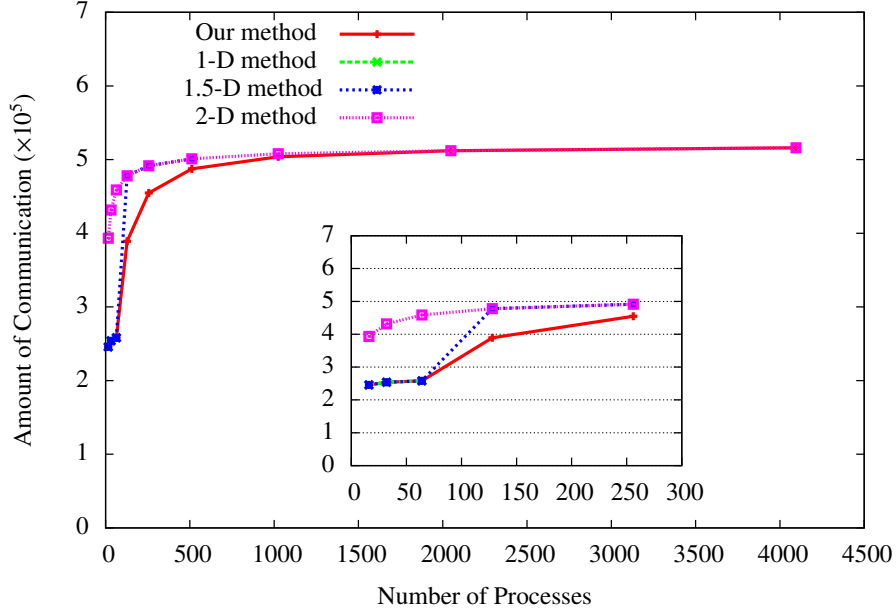
The gap between the best and worst orders is also higher with the 5-D FFT, especially with a small number of processes. With up to N^2 processes, it is exactly 4 times. The gap is from 2.1 to 2.7 times for the range of $[N^2, N^3]$, and up to 1.4 times for the next range of $[N^3, N^4]$. Similar to the 3-D and 4-D FFTs, the gap is found to be almost completely independent of N .

Based on these observations, our decomposition method for higher dimensional FFTs is thought to produce higher number of transpose orders that are always best for every range of the number of processes. And so is the gap in the amount of communication between the worst and best orders. Regarding the transpose order, we notice that the best orders of the 3-D FFT are in the form of either 1+2 or 2+1, i.e., $a(bc)$ or $(ab)c$, where 2 simply means a transpose $ab \rightarrow ba$. The form of $a(bc)$ leads to the subsequent transpose order of $a(cb)$, and then $(cb)a$, being one of the four best orders for the 3-D FFT. Likewise, the form of $(ab)c$ and its consequent orders of $c(ab)$ and $c(ba)$ are another best order. For the 4-D FFT, they follow the form of 2+2, $(ab)(cd)$, but neither 1+3, $a(bcd)$, nor 3+1, $(abc)d$. Meanwhile, the forms of 2+3, $(ab)(cde)$, and 3+2, $(abc)(de)$, are the best combinations for the 5-D FFT. Consequently, we expect the form of 3+3, $(abc)(def)$, to deliver better, if not best, performance for the 6-D FFT. Better forms for higher-dimensional FFTs can be derived in the same way such that their two parts are the most balanced.

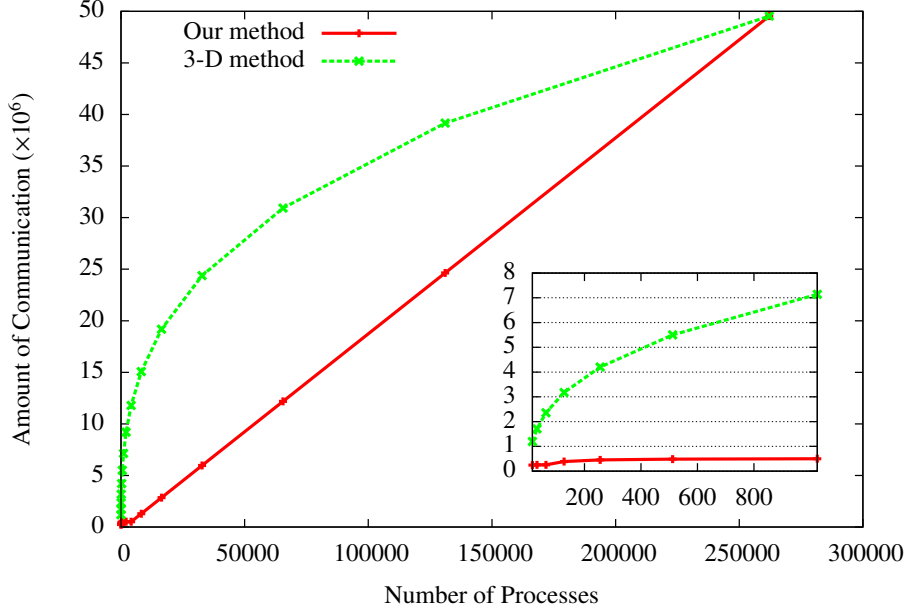
4. Comparison of Communication Amount

Figure 6 compares our proposed method with the 1-D method [10], 1.5-D method [28], and 2-D method [16] (a), and with the 3-D method [11] (b), for the 3-D FFT. The 3-D method is displayed in a separate figure for a clear presentation, as the amount of communication in this case is far larger than the other cases. The amount of communication is calculated in case of $64 \times 64 \times 64$ data points. Certainly, similar results can be produced for smaller and larger numbers of data points.

The 1-D method divides the 3-D data into blocks of equal numbers of complete ab -planes along the c -axis, and then allocates them to the processes to perform the 1-D FFTs along the a - and b -axes, followed by a data transpose



(a) Our method and the 1-D method [10], 1.5-D method [28], and 2-D method [16].



(b) Our method and the 3-D method [11].

Figure 6: Comparison for 3-D FFTs in terms of communication amount with $64 \times 64 \times 64$ data points.

so that each process contains complete ac -planes to carry out the 1-D FFTs along the remaining c -axis, with a total communication amount

$$A_{1D} = N^3 - \frac{N^3}{N_p}. \quad (21)$$

In the 2-D method, the ab -plane is evenly divided to the processes, with each having all the data along the remaining c -axis. The 1-D FFTs along c are executed, followed by two transpose steps so that the 1-D FFTs along the a - and b -axes can also be performed. The 2-D method has a communication amount

$$A_{2D} = 2N^3 - 2N_p \left(\frac{N^2}{N_p} \right)^{\frac{3}{2}}. \quad (22)$$

The 1.5-D method lies between these two methods, with the amount being similar to that of the 1-D method, when $N_p \leq N$, and the 2-D method, when $N < N_p \leq N^{1.5}$. Lastly, the 3-D method partitions the 3-D data along all three dimensions, and requires an amount

$$A_{3D} = 3N^3 \left(\frac{N}{\sqrt[3]{\frac{N^3}{N_p}}} - 1 \right). \quad (23)$$

As shown in Fig. 6(a), the 1-D method is able to work only along one dimension and is limited to 64 processes, while the 2-D method decomposes the domain in two dimensions, even for fewer than 64 processes. The 1.5-D method offers a compromise between the 1-D and 2-D methods. By contrast, our method is the most flexible and adaptive, as it partitions only along one dimension when the number of processes is up to 64 on condition that it is a divisor of 64, and decomposes in two dimensions while still starting from one dimension for a larger number of processes with transpose-order awareness to reuse as many data points as possible. As a result, up to 64 processes, our method works in the same fashion as the 1-D and 1.5-D methods provided that 64 is a multiple of the number of processes, and is about 60.0% to 77.8% better than the 2-D method with $64 \times 64 \times 64$ data points. From this point to 512 processes, the limit of the 1.5-D method, our method still has the edge over the 1.5-D and 2-D methods. Beyond the point of 512 processes, the 1.5-D method is no longer applicable, while the two other methods can operate until reaching the limit of $64 \times 64 = 4,096$ processes for the 2-D decomposition. The performance gap also gradually decreases, however,

and eventually becomes 0 from 2,048 processes. From 4,096 processes to the maximum 262,144 processes, only the 3-D decomposition is workable. Figure 6(b) demonstrates that our method outperforms the 3-D method, with the performance gain able to reach several orders of up to 11.6 (at 8,192 processes) for the 3-D decomposition alone.

Figure 7 extends the comparison by projecting the results obtained by our method and two other methods for the 4-D and 5-D FFTs, with $16 \times 16 \times 16 \times 16$ and $16 \times 16 \times 16 \times 16 \times 16$ data points, respectively, for a smaller number of processes. The 4-D and 5-D methods are assumed to operate in 4-D and 5-D decompositions, respectively, akin to the 3-D method. As a result, the amounts of communication are

$$A_{4D} = 4N^4 \left(\frac{N}{\sqrt[4]{\frac{N^4}{N_p}}} - 1 \right) \quad (24)$$

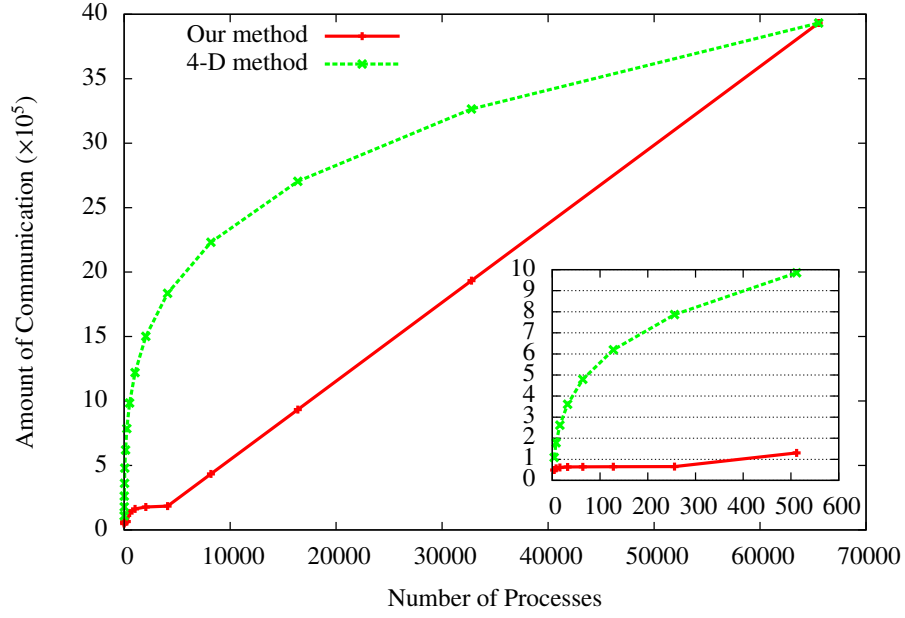
for the 4-D method, and

$$A_{5D} = 5N^5 \left(\frac{N}{\sqrt[5]{\frac{N^5}{N_p}}} - 1 \right) \quad (25)$$

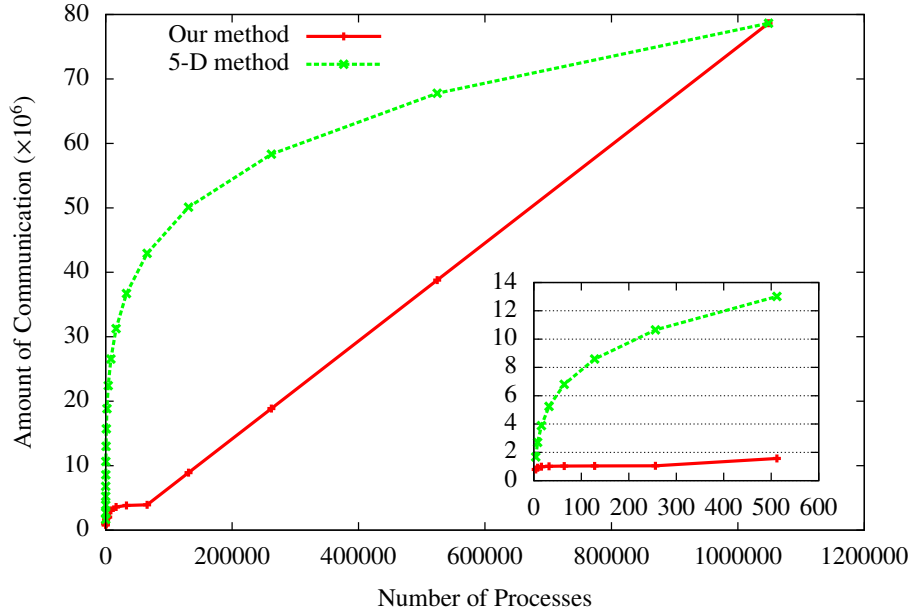
for the 5-D method. Given these conditions, our method has a distinct advantage, leaving a wide performance gap of up to approximately 12 times for the 4-D FFT, and 11.1 times for the 5-D FFT. The gap is found to remain almost unchanged by N .

5. Conclusion

We have presented our decomposition method for parallelization of multi-dimensional FFTs. The communication amount is the smallest compared to previously proposed methods, accomplished by adaptive decomposition and transpose order awareness. Featured by the row-wise decomposition that translates the M -D data into 1-D data and evenly divides the resultant 1-D data to the processes, our method can adaptively decompose the FFT data on the lowest possible dimensions based on the number of processes. In addition, our row-wise decomposition method provides a lot of alternatives in data transpose, among them the best communication efficient orders are identified and applied. We have determined the best transpose orders for the 3-D, 4-D, and 5-D FFTs, dependent on which we find out the way for



(a) 4-D FFTs with $16 \times 16 \times 16 \times 16$ data points.



(b) 5-D FFTs with $16 \times 16 \times 16 \times 16 \times 16$ data points.

Figure 7: Comparison for 4-D and 5-D FFTs.

deriving the transpose orders that can deliver better performance for higher-dimensional FFTs. Comparison in terms of communication amount shows that our method is superior to other methods for the 3-D FFT, and it is anticipated to have a distinct advantage for higher-dimensional FFTs. Actually, the method has been employed in our open-source density functional theory code called OpenMX [29]. Boosting communication efficiency while not sacrificing scalability, our method is promising to be harnessed in development of highly efficient parallel packages for multi-dimensional FFTs.

Acknowledgements

This work was supported by the Strategic Programs for Innovative Research (SPIRE), MEXT, and the Computational Materials Science Initiative (CMSI), and Materials Design through Computics: Complex Correlation and Non-Equilibrium Dynamics A Grant in Aid for Scientific Research on Innovative Areas, MEXT, Japan.

References

- [1] J. W. Cooley, J. W. Tukey, *Math. Comp.* 19 (1965) 297–301.
- [2] P. D. Haynes, M. Cote, *Comput. Phys. Commun.* 130 (2000) 130 – 136.
- [3] L. Clarke, I. Stich, M. Payne, *Comput. Phys. Commun.* 72 (1992) 14 – 28.
- [4] S. A. Broughton, K. M. Bryan, *Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing*, Wiley, 2008.
- [5] R. Gonzales, R. Woods, *Digital Image Processing*, Addison-Wesley Publishing Company, 1992.
- [6] D. Takahashi, A. Uno, M. Yokokawa, in: *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICSS)*, 2012 IEEE 14th International Conference on, pp. 344 –350.
- [7] D. Takahashi, in: *Proceedings of the 8th international conference on Applied parallel computing: state of the art in scientific computing, PARA’06*, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 1178–1187.

- [8] R. Al Na'mneh, D. Pan, R. Adhami, in: System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on, pp. 307 – 311.
- [9] M. Frigo, S. Johnson, Proceedings of the IEEE 93 (2005) 216 –231.
- [10] P. Dmitruk, L.-P. Wang, W. Matthaeus, R. Zhang, D. Seckel, Parallel Comput. 27 (2001) 1921 – 1936.
- [11] M. Eleftheriou, J. Moreira, B. Fitch, R. Germain, in: T. Pinkston, V. Prasanna (Eds.), High Performance Computing - HiPC 2003, volume 2913 of *Lect. Notes Comput. Sc.*, Springer Berlin Heidelberg, 2003, pp. 194–203.
- [12] M. Eleftheriou, B. Fitch, A. Rayshubskiy, T. Ward, R. Germain, in: J. Cunha, P. Medeiros (Eds.), Euro-Par 2005 Parallel Processing, volume 3648 of *Lect. Notes Comput. Sc.*, Springer Berlin Heidelberg, 2005, pp. 795–803.
- [13] M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, T. J. C. Ward, R. S. Germain, IBM J. Res. Dev. 49 (2005) 457 –464.
- [14] B. Fang, Y. Deng, G. Martyna 176 (2007) 531 – 538.
- [15] O. Ayala, L.-P. Wang, Parallel Computing 39 (2013) 58 – 77.
- [16] D. Takahashi, in: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski (Eds.), Parallel Processing and Applied Mathematics, volume 6067 of *Lect. Notes Comput. Sc.*, Springer Berlin / Heidelberg, 2010, pp. 606–614.
- [17] D. Pekurovsky, in: San Diego Super Computer Summer Institute, <http://code.google.com/p/p3dfft/>, 2007.
- [18] N. Li, S. Laizet, in: Cray User Group 2010 Conference, www.2decomp.org, 2010.
- [19] N. Cardoso, P. J. Silva, P. Bicudo, O. Oliveira, Comput. Phys. Commun. 184 (2013) 124 – 129.
- [20] A. Eklund, M. Andersson, H. Knutsson, Int. J. Biomed. Imaging 2011 (2011) 16 pages.

- [21] A. Veeraraghavan, R. Raskar, A. Agrawal, A. Mohan, J. Tumblin, ACM Trans. Graph. 26 (2007).
- [22] G. Wetzstein, I. Ihrke, W. Heidrich, Int. J. Comput. Vision (2012) 1–17.
- [23] D. W. Ritchie, D. Kozakov, S. Vajda, Bioinformatics 24 (2008) 1865–1873.
- [24] D. W. Ritchie, V. Venkatraman, L. Mavridis, in: T. Solomonides, I. Blanquer, V. Breton, T. Glatard, Y. Legr (Eds.), Healthgrid Applications and Core Technologies, volume 159 of *Studies in Health Technology and Informatics*, IOS Press, 2010, pp. 146–155.
- [25] D. W. Ritchie, in: T. C. Lee Banting (Ed.), Drug Design Strategies Computational Techniques and Applications, RSC Drug Discovery, The Royal Society of Chemistry, 2012, pp. 56–86.
- [26] J. A. Kovacs, P. Chacón, Y. Cong, E. Metwally, W. Wriggers, Acta Crystallogr. Sect. D 59 (2003) 1371–1376.
- [27] T. V. T. Duy, T. Ozaki, arXiv:1209.4506 (2012) 43 pages.
- [28] D. Takahashi, A. Yee, T. Hoefer, C. Coti, J. Kim, F. Cappello, in: The seventh workshop of the INRIA-Illinois-ANL Joint Laboratory on Petascale Computing, 2012.
- [29] OpenMX, Open source package for Material eXplorer, <http://www.openmx-square.org/>, retrieved 2013-01-15.